EXHIBIT 9

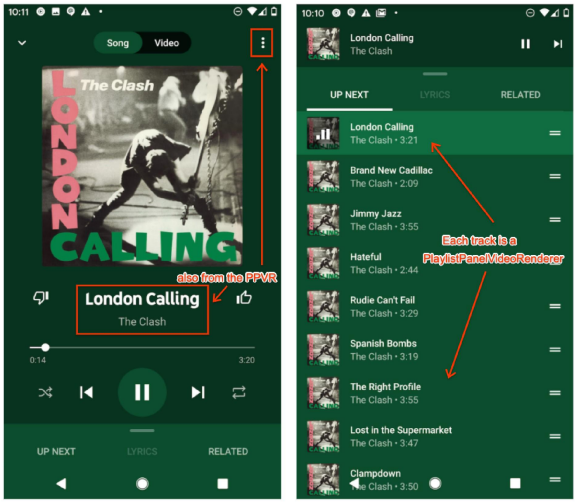
FILED UNDER SEAL

The Queue

Reviewed by davidjn on 2020-08-19 · Edited 2020-09-30

Overview

The YouTube Music queue is primarily a client-side construct. YouTube Music 1st party clients persist a list of tracks represented by `PlaylistPanelVideoRenderer` messages. These renderers are served through GetMusicWatchNext when users initiate a new queue and GetMusicQueue when users "Add to queue" or "Play next" via in-app context menus. All other queue actions, such as removing items, re-ordering, or shuffle/repeat are performed as manipulations on the client-persisted list of renderers.



When a song from the queue is currently playing, some data from the same `PlaylistPanelVideoRenderer` message used for dislay in the queue is re-purposed for use in the primary watch page UI.

Notable exceptions to the above are:

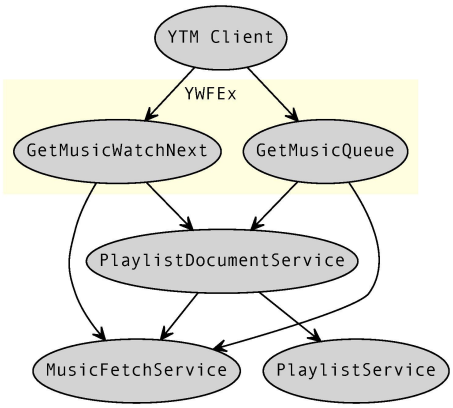
- When Casting, the queue is persisted as a server-side "remote queue"
- A/V Switching, where `PlaylistPanelVideoRenderer` messages are replaced with equivalent `PlaylistPanelVideoWrapperRenderer` messages.

Implementation

Server

1st party

The primary components involved in serving the queue for 1st party clients are diagrammed below.



YouTube Music Clients make requests to two endpoints, GetMusicWatchNext and GetMusicQueue, to obtain renderers for displaying the queue to the user.

The **GetMusicWatchNext** endpoint provides queue renderers when initiating playback on a new container, on queue continuation loads, and for the autoplay section of the queue.

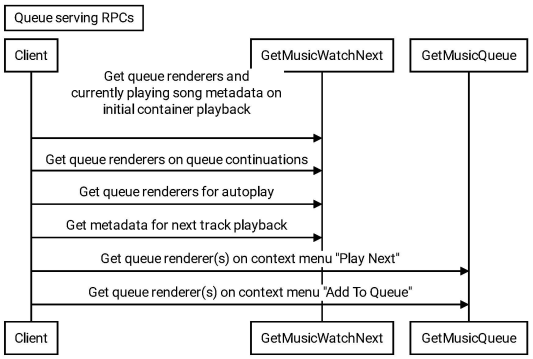
The **GetMusicQueue** endpoint provides queue renderers when users perform "Add to Queue" or "Play Next" actions from in-app context menus.

The **PlaylistDocumentService** (PLDS) provides data needed to serve playlists. It is used in the music queue to load the containing playlist for the content that the user initiated playback on. It loads playlist data via the PlaylistService and decorates with music-specific data via the MusicFetchService.

The **MusicFetchService** provides music-specific metadata for individual tracks in the queue. It's FetchedTrack content is served through the PlaylistDocumentService for playlist-based queue operations, and directly through GetMusicWatchNext and GetMusicQueue for track-based queue operations.

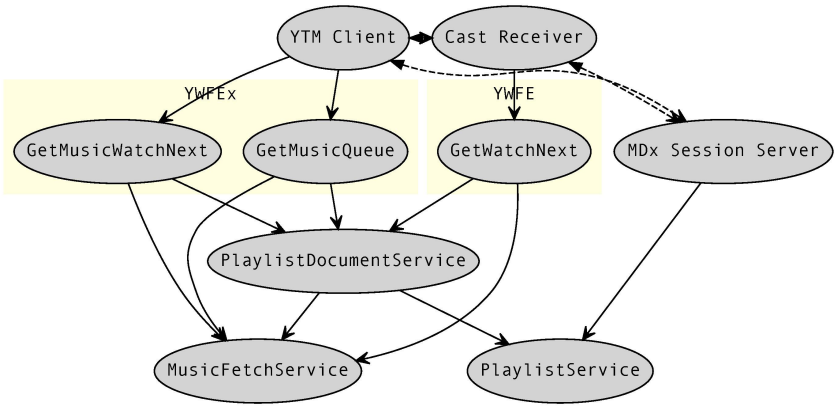
The **PlaylistService** interfaces directly with playlist storage systems and provides base playlist data, such as which video IDs are in a given playlist.

Note that all dependencies are read only dependencies. Different client-side queue actions result in calls to one or the other of `GetMusicWatchNext` or `GetMusicQueue`:



Cast

YouTube Music clients can Cast to Living Room devices. In comparison to the 1st party case where the queue is a client-side construct, the Casting use case stores the queue in YouTube servers as a "Remote Queue" playlist. The architecture diagram looks like:



New components in this diagram relative to the 1st party case are the `Cast Receiver` device, the `MDx Session Server`, and a `GetWatchNext` endpoint that behaves similarly to `GetMusicWatchNext`. Furthermore, the `GetWatchNext`, `GetMusicWatchNext`, and `GetMusicQueue` endpoints have been in grouped in their respective containing binaries. Notably, the Cast devices are served through the YWFE, while YouTube Music devices are served through the YWFEx.

The **Cast Receiver device** is the Cast-enabled device that the user connects. This device is the where actual audio playback occurs.

The **MDx Session Server** manages the "Remote Queue" playlist as well as the broader multi-device experience while Casting. Clients make queue edit operations (add to queue, re-order, remove from queue, etc) through the MDx Session Server, which makes appropriate calls to the PlaylistService to edit the underlying "Remote Queue" playlist and also pushes updates to connected client devices to notify that the queue has been modified. Clients (both the 1st party and Cast receiver devices) then obtain renderers for display via read-only InnerTube endpoints as described in the 1st party use case.

The **GetWatchNext** endpoint serves metadata about the queue and currently playing track to living room devices. It behaves similarly to the GetMusicWatchNext endpoint. It shares most of the same code for orchestrating RPCs and serving metadata.

See here for more information.

Android

See this doc for the most recent deep dive on the YouTube Music queue implementation.

Queue Persistence

YouTube Music clients persist the queue when the application is closed so that users can continue their queue upon re-opening the app.

Here are some links to relevant docs:

iOS

- YTM Queue Restoration (iOS). Describes the design.
- YTM iOS Extending Queue Persistence. Experiment doc for extending coverage on queue persistence.

Android

- go/ytm-playback-persistence. Describes the previous design and some outdated proposals.
- go/ytm-persistence-refactor. Describes the refactored design.
- go/ytm-queue-persistence-refactor-experiment. Experiment doc for launching the refactor.

Historical music queue docs

- 2017. go/androidqueueapi. Queue library implementation on Android.
- 2017. go/orbit-queue-android-impl. Support for music queue Cast integration.
- 2018. go/ngwqueuedesign2. Client-side queue navigation design on Android.
- 2019. go/ytm-queue-ui. Decoupling the queue UI from an internal queue representation on Android.
- 2020. go/android-queue-research. Includes an detailed overview of the YTM queue implementation on Android.

Was this page helpful?

Comments

Before you can create or read g3doc comments, you need to grant g3doc access to Buganizer and Google people information. Learn more

Get access to g3doc comments

View all unresolved or resolved comments

View Markdown source · Edit this page · Project · File a documentation bug · Incoming links · Recent CLs · Served by g3doc